APPENDIX G

Uucp Administrator's Manual

CONTENTS

Uucp Administrator's Manual

1. Introduction

Uucp has been the mainstay of UNIX[1] system to UNIX system communication for several years [1, 2]. This document illustrates how a network is set up, the format of control files and administrative procedures. Administrators should be familiar with the *Uucp Users Tutorial* and the manual pages for each of the *uucp* related commands before reading this document.

2. Planning

In setting up a network of UNIX systems there are several considerations that should be taken into account *before* configuring each system on the network. The following sections attempt to outline the most important considerations.

2.1 Extent of the Network

Some basic decisions about *access* to processors in the network must be made before attempting to set up the configuration files. If an administrator has control over only one processor and an existing network is being joined, then the administrator must decide what level of access should be granted to other systems. The other members of the network must make a similar decision for the new system. The UNIX system *password* mechanism is used to grant access to other systems. The file *(/usr/lib/uucp/USERFILE)* restricts access by other systems to parts of the filesystem tree and the file */usr/lib/uucp/L.sys* on the local processor determines how many other systems on the network can be reached.

When setting up more than one processor is involved, the administrator has control of a larger fraction of the network and can make more decisions about the setup of the network. For example, the network can be set up as a *private* network where only those machines under the direct control of the-administrator can access each other. Granting *no* access to machines outside the network can be done if security is paramount, however, this is usually impractical. Very limited access can be granted to outside machines by each of the systems on the *private* network. Alternatively, access to/from the outside world can be confined to only one processor. This is frequently done to minimize the effort in keeping access information (passwords, phone numbers, login sequences, etc.) updated and to minimize the number of security holes for the private network.

2.2 Hardware and Line Speeds

There are only two supported means of interconnection by *uucp(1)*,

1. Direct connection using a null modem.
2. Connection over the DDD network.

Direct connection over private lines using X.25 is not fully supported in UNIX System V, although with the addition of one program *x25.login* it can be made operational. In choosing hardware, the equipment used by other processors on the network must be considered. For example, if some systems on the network have only 103 type (300 baud) datasets, then communication with them is not possible unless the local system has a 300 baud dataset connected to a calling unit. (Most datasets available on systems are 1200 baud.) If hardwired connections are to be used between systems, then the *distance* between systems must be considered since a null modem cannot be used when the systems are separated by more than several hundred feet (the limit for communication at

* UNIX is a Trademark of Bell Telephone Laboratories, Incorporated.

9600 baud is about 800 to 1000 feet although the RS232 specification allows for less than fifty feet). Limited distance modems must be used beyond these distances or if noise on the lines becomes a problem.

2.3 Maintenance and Administration

There is a minimum amount of maintenance that must be provided on each system to keep the access files updated, to insure that the network is running properly and to track down line problems. When more than one system is involved, the job becomes more difficult because there are more files to update and because users are much less patient when failures occur between machines that are under local control.

3. A Quick Tour of the Uucp Software

Figure 1 is an illustration of the daemons used by the *uucp* network to communicate with another system. The *uucp(1)* or *uux(1)* command queues users requests and spawns the *uucico* daemon to call another system. Figure 2 illustrates the structure of *uucico* and the tasks that it performs in communicating with another system. It initiates the call to another system and performs the file transfer. On the receiving side, *uucico* is invoked to receive the transfer. Remote execution jobs are actually done by transferring a command file to the remote system and invoking a daemon *(uuxqt)* to execute that command file and return the results.

4. Installation

The *uucp(1)* package is delivered as part of the standard UNIX system distribution. It resides in its own subdirectory (called *uucp)* in the commands area and has its own make file *(uucp.mk)*. The *uucp* package is installed as part of the normal distribution, however, if it must be reinstalled for any reason, then the sequence

```
make uucp.mk install
```

should be executed.

4.1 Object Modules

The following object modules are installed as part of the *uucp* make procedure,

1. *Uucp* - The file transfer command.
2. *Uux* - The remote execution command.
3. *Uucico* - The *uucp* network daemon.
4. *Uustat* - Network status command.
5. *Uuclean* - Cleanup command.
6. *Uusub* - The command for monitoring and creating a subnetwork.
7. *Uuxqt* - The remote execution daemon.
8. *Uudemon.day* - A shell procedure that is invoked each day to maintain the network. Shell scripts for execution each week *(uudemon.wk)* and each hour *(uudemon.hr)* are also distributed.

4.2 Password FIle

To allow remote systems to call the local system, password entries must be made for any *uucp* logins. For example,

```
nuucp:zaaAA:6:1:45422-UUCP.Admin:/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

Note that the *uucico* daemon is used for the *shell* and the spool directory is used as the working directory.

4.3 Lines File

The file */usr/lib/uucp/L-devices* contains the list of all lines that are directly connected to other systems or are available for calling other systems. The file contains the attributes of the lines and whether the line is a permanent connection or can call via a dialer. The format of the file is

type line call-device speed protocol

where each field is

type	Two keywords are used to describe whether a line is directly connected to another system (DIR) or uses an automatic calling unit (ACU). An X.25 permanent virtual circuit would use the DIR keyword.
line	This is the device name for the line (e.g., *ttyab* for a direct line, *cul0* for a line connected to an ACU).
call-device	If the ACU keyword is specified, this field contains the device name of the automatic calling unit. Otherwise, the field is ignored, however, a placeholder must be used in this field so that the *protocol* field can be interpreted.
speed	The line speed that the connection is to run at. (The speed field is currently ignored if an X.25 link is used.)
protocol	This is an optional field that needs only be filled in if the connection is for a protocol other than the default terminal protocol. The X.25 protocol is the only other protocol supported and the single character *x* is used to select this protocol.

The following entries illustrate various types of connections,

```
DIR ttyab 0 9600
ACU cul0 cua0 1200
DIR x25.s0 0 300 x
```

The first entry is for a hardwired line running at 9600 baud between two systems. Note that the *acu-device* field is zero. The second entry is for an line with a 1200 baud automatic calling unit. The last entry is for an X.25 synchronous direct connection between systems. Note that the *protocol* field is filled in and that the *acu-device* and *line speed* fields are meaningless.

4.3.1 Naming Conventions It is often useful when naming lines that are directly connected between systems or which are dedicated to calling other systems to choose a naming scheme that conveys the use of the line. In the earlier examples, the name *ttyab* is used for the line that directly connects two systems named *a* and *b*. Similarly, lines associated with calling units are best given names that relate them to the their calling unit (note the names *cul0* and *cua0* to specify the line and calling unit respectively).

4.4 System File

Each entry in this file represents a system that can be called by the local *uucp* programs. More than one line may be present for a particular system. In this case, the additional lines represent alternative communication paths that will be tried in sequential order. The fields are described below.

system name The name of the remote system.

time This is a string that indicates the days-of-week and times-of-day when the system should be called (e.g., MoTuTh0800−1730).

The day portion may be a list containing some of *Su Mo Tu We Th Fr Sa* or it may be *Wk* for any week-day or *Any* for any day. The time should be a range of times (e.g., 0800−1230). If no time portion is specified, any time of day is assumed to be allowed for the call. Note that a time range that spans 0000 is permitted, for example, 0800-0600 means all times are allowed other than times between 6 and 8 am. An optional subfield is available to specify the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a "," (e.g., Any,9 means call any time but wait at least 9 minutes before retrying the call after a failure has occurred).

device This is either *ACU* or the hard-wired device name to be used for the call. For the hard-wired case, the last part of the special file name is used (e.g., tty0).

class This is usually the line speed for the call (e.g., 300).

phone The phone number is made up of an optional alphabetic abbreviation (dialing prefix) and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (e.g., mh5900, boston995−9980). For the hard-wired devices, this field contains the same string as used for the *device* field (e.g., tty0, etc.).

login The login information is given as a series of fields and subfields in the format

```
[ expect  send ] ...
```

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The expect field may be made up of subfields of the form

```
expect[−send−expect] ...
```

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. (e.g., login--login will expect *login*; if it gets it, the program will go on to the next field; if it does not get *login*, it will send *null* followed by a new line, then expect *login* again.)

There are two special names available to be sent during the login sequence. The string *EOT* will send an EOT character and the string *BREAK* will try to send a *BREAK* character. (The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems.) A number from 1 to 9 may follow the *BREAK* for example, *BREAK1*, will will send 1 null character instead of the default of 3. Note that *BREAK1* usually works best for 300/1200 baud lines.

A typical entry in the L.sys file would be

```
sys Any ACU 300 mh7654 login uucp ssword: word
```

The expect algorithm matches all or part of the input string as illustrated in the password field above.

4.5 Dialing Prefixes

This file contains the dial-code abbreviations used in the *L.sys* file (e.g., py, mh, boston). The entry format is

```
abb  dial-seq
```

where

abb	is the abbreviation,
dial-seq	is the dial sequence to call that location.

The line

```
py  165−
```

would be set up so that entry py7777 would send 165−7777 to the dial-unit.

4.6 USERFILE

This file contains user accessibility information. It specifies four types of constraint,

[1] which files can be accessed by a normal user of the local machine,

[2] which files can be accessed from a remote computer,

[3] which login name is used by a particular remote computer,

[4] whether a remote computer should be called back in order to confirm its identity.

Each line in the file has the format

```
login,sys  [ c ]  path-name  [ path-name ] ...
```

where

login	is the login name for a user or the remote computer,
sys	is the system name for a remote computer,
c	is the optional *call-back required* flag,
path-name	is a path-name prefix that is acceptable for *sys*.

The constraints are implemented as follows.

[1] When the program is obeying a command stored on the local machine, the path-names allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.

[2] When the program is responding to a command from a remote machine, the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.

[3] When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE*. There may be several lines with the same login name but one of

them must either have the name of the remote system or must contain a *null* system name.

[4] If the line matched in ([3]) contains a "c", the remote machine is called back before any transactions take place.

The line

```
        u,m  /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with "/usr/xyz". The line

```
    you,  /usr/you
```

allows the ordinary user *you* to issue commands for files whose name starts with "/usr/you". (Note that this type restriction is seldom used.) The lines

```
    u,m /usr/xyz  /usr/spool
    u,  /usr/spool
```

allows *any* remote machine to login with name *u*. If its system name is not *m*, it can only ask to transfer files whose names start with "/usr/spool". If it is system *m*, it can send files from paths "/usr/xyz" as well as "/usr/spool". The lines

```
    root,  /
    ,  /usr
```

allow any user to transfer files beginning with "/usr" but the user with login *root* can transfer any file. (Note that any file that is to be transferred must be readable by anybody.)

4.7 Forwarding File

There are two files that allow restrictions to be placed on the forwarding mechanism. The format of the entries in each file is the same,

```
        system
```

or

```
        system!user!user2,...
```

The file *ORIGFILE (/usr/lib/uucp/ORIGFILE)* restricts the access of systems that are attempting to forward *through the local system.* The file contains the list of systems (and users) for whom the local system is willing to forward. Each entry refers to the system that was the *source* of the original job and not the name of the last system to forward the file. The second file *FWDFILE (/usr/lib/uucp/FWDFILE)* is a list of valid systems that a job can be *forwarded to* (it is not necessarily the name of the destination of a job, but merely the next valid node). This file will be a subset of the L.sys file and can be used to prevent forwarding to systems that are very expensive to reach, but to which access by local users is allowed (for example, links to overseas universities). If neither of these files exist, *uucp* will be perfectly happy to forward for any system. As an example, if the entry for system *australia* were in the ORIGFILE but not in the FWDFILE on system xnode, it would mean that system *australia* would be capable of forwarding jobs into the network via system xnode however, no systems in the network could forward a job to *australia* via system xnode.

5. Administration

The role of the *uucp* administrator depends heavily on the amount of traffic that enters or leaves a system and the quality of the connections that can be made to and from that system. For the average system, only a modest amount of traffic (100-200 files per day) pass through the system and little if any intervention with the *uucp* automatic cleanup functions is necessary. Systems that pass large numbers of files (200-10000) may require more attention when problems occur. The following sections describe the routine administrative tasks that must be performed by the administrator or are automatically performed by the *uucp* package. The section on problems describes what are the most frequent problems and how to effectively deal with them.

5.1 Cleanup

The biggest problem in a dialup network like *uucp* is dealing with the backlog of jobs that cannot be transmitted to other systems. The following cleanup activities should be routinely performed by shell scripts started from *cron(1)*.

5.1.1 Cleanup of Undeliverable Jobs The *uudemon.day* procedure usually contains an invokation of the *uuclean* commany to purge any jobs that are older than some fixed time (usually 72 hours). A similar procedure is usually used to purge any *lock* or *status* files. An example invokation of *uuclean(1m)* to remove both job files and old status files every 48 hours is

```
/usr/lib/uuclean -pST -pC -n48
```

5.1.2 Cleanup of the Public Area In order to keep the local filesystem from overflowing when files are sent to the public area, the *uudemon.day* procedure is usually set up with a *find* command to remove any files that are older than seven days. This interval may need to be shortened if there is not sufficient space to devote to the public area.

5.1.3 Compaction of Log Files The files *SYSLOG* and *LOGFILE* that contain logging information are compacted daily (using the *pack* command) and should be kept for one week before being overwritten.

5.2 Polling Other Systems

Systems that are passive members of the network must be polled by other systems in order for their files to be sent. This can be arranged by using the *uusub(1)* command as follows,

```
uusub -ccnode
```

which will call *cnode* when it is invoked.

5.3 Problems

The following sections list the most frequent problems that appear on systems that make heavy use of *uucp(1)*.

5.3.1 Out of Space The filesystem used to spool incoming or outgoing jobs can run out of space and prevent jobs from being spawned or much worse received from remote systems. The inability to receive jobs is the worse of the two conditions since when filespace does become available, the system will be *flooded* with the backlog of traffic.

5.3.2 Bad Acu's and Modems The automatic calling units and incoming modems occasionally cause problems that make it difficult to contact other systems or to receive files. These problems are usually readily identifiable since *LOGFILE* entries will usually point to the bad line. If a bad line is suspected, it it useful to use the *cu(1)* command to try calling another system using the suspected line.

5.3.3 Administrative problems Some *uucp* networks have so many members that it is difficult to keep track of changing passwords, changing phone numbers or changing logins on remote systems. This can be a very costly problem since acu's will be tied up calling a system that cannot be reached.

6. Debugging

In order to verify that a system on the network can be contacted, the *uucico* daemon can be invoked from a user's terminal directly. For example, to verify that *cnode* can be contacted, a job would be queued for that system as follows,

```
uucp -r file cnode!~/tom
```

The *-r* option forces the job to be queued but does not invoke the daemon to process the job. The *uucico* command can then be invoked directly,

```
/usr/lib/uucp/uucico -r1 -x4 -scnode
```

The *-r1* option is necessary to indicate that the daemon is to start up in *master* mode (that is, it is the calling system). The *-x4* specifies the level of debugging that is to be printed. Higher levels of debugging can be printed (greater than 4) but requires familiarity with the internals of *uucico.* If several jobs are queued for the remote system, it is not possible to force *uucico* to send one particular job first. The contents of LOGFILE should also be monitored for any error indications that it posts. Frequently, problems can be isolated by examining the entries in LOGFILE associated with a particular system. The file *ERRLOG* also contains error indications.

7. Conclusion

This manual has emphasized the format of control files and some of the issues in setting up a *uucp* network.

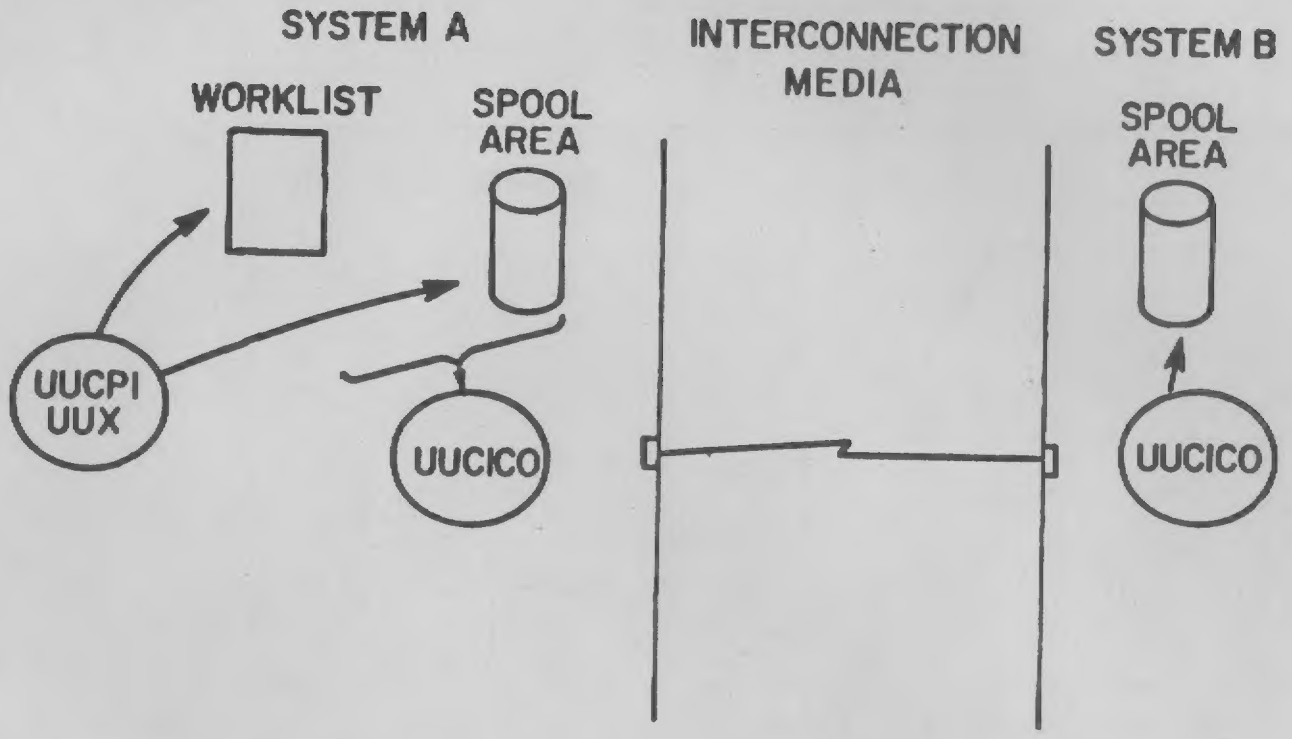

Figure 1 Uucp network daemons.

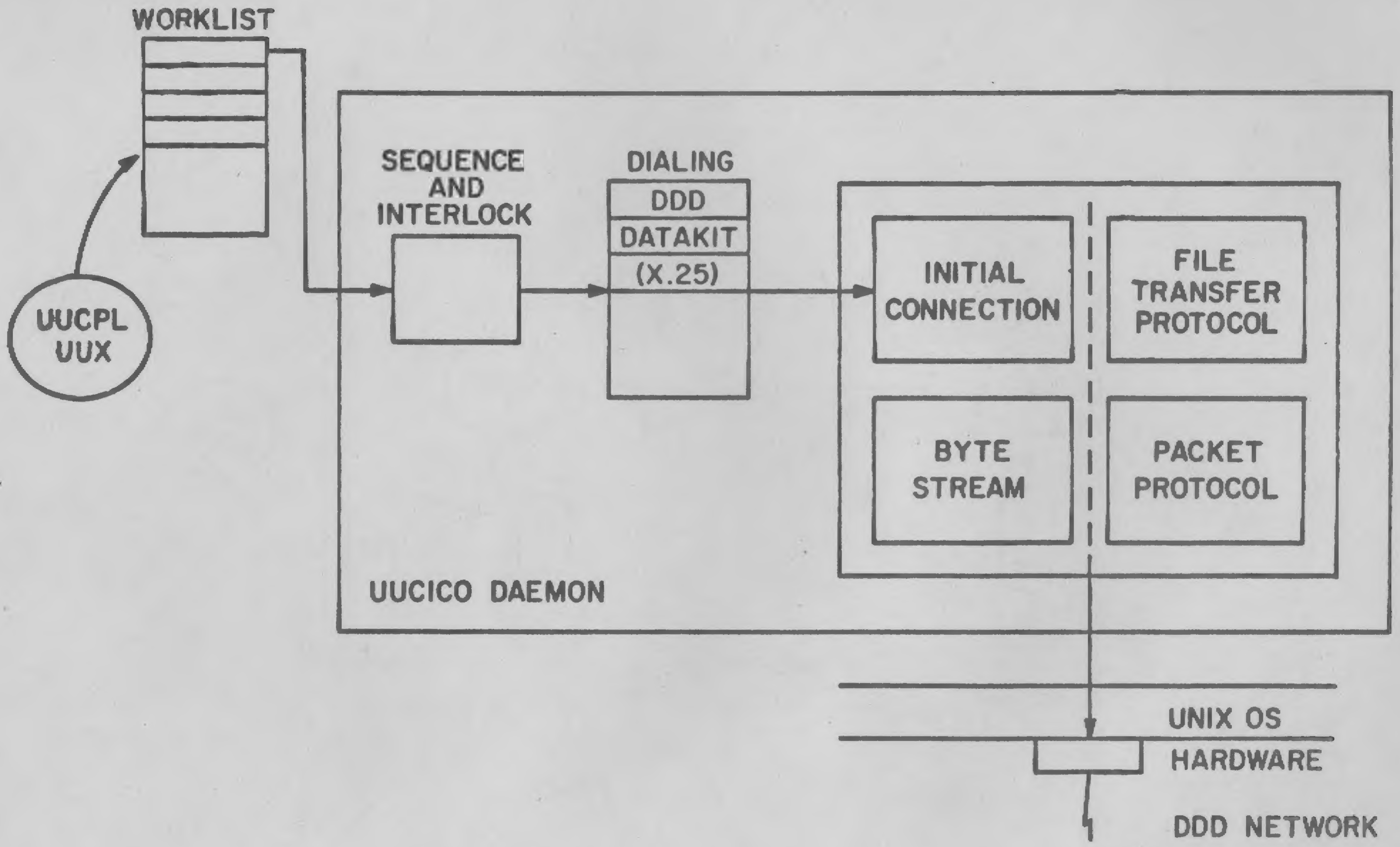

Figure 2 Uucico daemon functional blocks.